université de BORDEAUX

Formation CNRS
18 Novembre 2016
Python pour la biologie

Biopython: BLAST

cgfb
BIOINFORMATIQUE

université de BORDEAUX

Qu’est ce que Blast ?

- The Biopython Project is an international association of developers of freely available Python tools for computational molecular biology

- The Biopython web site (http://www.biopython.org) provides an online resource for modules, scripts, and web links for developers of Python-based software for bioinformatics use and research.

- Basically, the goal of Biopython is to make it as easy as possible to use Python for bioinformatics by creating high-quality, reusable modules and classe

```
>>> from Bio.Blast import NCBIWWW
>>> help(NCBIWWW.qblast)
```

Running BLAST over the Internet

- search against the nucleotide database (nt)

```
>>> from Bio.Blast import NCBIWWW
>>> result_handle = NCBIWWW.qblast("blastn", "nt", "8332116")
```

- sequence already in a FASTA formatted file

```
>>> from Bio.Blast import NCBIWWW
>>> fasta_string = open("m_cold.fasta").read()
>>> result_handle = NCBIWWW.qblast("blastn", "nt", fasta_string)
```

- read in the FASTA file as a SeqRecord

```
>>> from Bio.Blast import NCBIWWW
>>> from Bio import SeqIO
>>> record = SeqIO.read("m_cold.fasta", format="fasta")
>>> result_handle = NCBIWWW.qblast("blastn", "nt", record.seq)
```

- Supplying just the sequence means that BLAST will assign an identifier

```
>>> from Bio.Blast import NCBIWWW
>>> from Bio import SeqIO
>>> record = SeqIO.read("m_cold.fasta", format="fasta")
>>> result_handle = NCBIWWW.qblast("blastn", "nt", record.format("fasta"))
```

Running BLAST over the Internet

- Get back your results in a handle object (by default in XML format).
- Next step would be to parse the XML output into Python objects representing the search result
- save a local copy of the output le first
- Use “result_handle.read()” to read the BLAST output only once

```
>>> save_file = open("my_blast.xml", "w")
>>> save_file.write(result_handle.read())
>>> save_file.close()
>>> result_handle.close()
```

- just open the saved file for input:

```
>>> result_handle = open("my_blast.xml”)
```

- Several format as XML, HTML, and plain text

```
>>> from Bio.Blast import NCBIWWW
>>> result_handle = NCBIWWW.qblast("blastn", "nt", "8332116")
```

- have the BLAST output (in XML format) in the file “my_blast.xml”

```
>>> result_handle = open("my_blast.xml")
```

Parsing BLAST output

- single BLAST result (single query sequence)

```
>>> from Bio.Blast import NCBIXML
>>> blast_records = NCBIXML.read(result_handle)
```

- Lots of results (multiple query sequences)

```
>>> blast_records = NCBIXML.parse(result_handle)
```

- Like Bio.SeqIO and Bio.AlignIO, we have a pair of input functions, read (one object) and parse (multiple objects)
- NCBIXML.parse() returns an iterator.

```
>>> blast_records = NCBIXML.parse(result_handle)
>>> blast_record = next(blast_records)
# ... do something with blast_record
>>> blast_record = next(blast_records)
# ... do something with blast_record
>>> blast_record = next(blast_records)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
StopIteration
# No further records
```

Parsing BLAST output (2)

- Or, you can use a for loop

```
>>> for blast_record in blast_records:
... # Do something with blast_record
```

- Step through the BLAST records only once.
- If you want to save all returned BLAST records, you can convert the iterator into a list

```
>>> blast_records = list(blast_records)
```

- Usually, you'll be running one BLAST search at a time

```
>>> from Bio.Blast import NCBIXML
>>> blast_records = NCBIXML.parse(result_handle)
>>> blast_record = next(blast_records)
```

- or more elegantly

```
>>> from Bio.Blast import NCBIXML
>>> blast_record = NCBIXML.read(result_handle)
```

The BLAST record class (1)

- Everything you might ever want to extract from the BLAST output

```
>>> E_VALUE_THRESH = 0.04
>>> for alignment in blast_record.alignments:
... for hsp in alignment.hsps:
... if hsp.expect < E_VALUE_THRESH:
... print('****Alignment****')
... print('sequence:', alignment.title)
... print('length:', alignment.length)
... print('e value:', hsp.expect)
... print(hsp.query[0:75] + '...')
... print(hsp.match[0:75] + '...')
... print(hsp.sbjct[0:75] + '...')
```

```
****Alignment****
sequence: >gb|AF283004.1|AF283004 Arabidopsis thaliana cold acclimation protein WCOR413-like protein
alpha form mRNA, complete cds
length: 783
e value: 0.034
tacttgttgatattggatcgaacaaactggagaaccaacatgctcacgtcacttttagtcccttacatattcctc...
|||||||||| | |||||||||||| || |||| || || |||||||| |||||| | | |||||||| ||| ||...
tacttgttggtgttggatcgaaccaattggaagacgaatatgctcacatcacttctcattccttacatcttcttc...
```

- Parsers return Record objects
- These objects are defined in Bio.Blast.Record and are quite complete
- http://biopython.org/DIST/docs/tutorial/Tutorial.pdf#section.7.4